Exhibit B

SYNCHRONIZER.H

```
#if
!defined(AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E
0__INCLUDED_)
#define
AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E0__INC
LUDED_

// SYNCHRONIZER.H - Header file for your Internet Server
//    Synchronizer Extension

#include "resource.h"

class isapithread;

class CSynchronizerExtension : public CHttpServer
{
protected:
        isapithread *    m_pisapithread;
public:
        CSynchronizerExtension();
        ~CSynchronizerExtension();

// Overrides
        // ClassWizard generated virtual function overrides
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //{{AFX_VIRTUAL(CSynchronizerExtension)
        public:
        virtual BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
        //}}AFX_VIRTUAL
        virtual BOOL TerminateExtension(DWORD dwFlags);

        // TODO: Add handlers for your commands here.
        // For example:

        void Default(CHttpServerContext* pCtxt);

        DECLARE_PARSE_MAP()

        void BroadcastVideoEvents(CHttpServerContext* pCtxt, LPTSTR lpUserName,
LPCTSTR lpDiskId, long nLocationID, LPCTSTR lpBCA,LPCTSTR lpProcessorID,
LPCTSTR lpDecoderID);
        void ServerInfo(CHttpServerContext* pCtxt, LPCTSTR lpDiskId, LPCTSTR
nLocationID);
```

```
        //{{AFX_MSG(CSynchronizerExtension)
        //}}AFX_MSG
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_SYNCHRONIZER_H__B20D9954_E973_4C6A_AAA6_A5F95F45A7E
0__INCLUDED)
```

SYNCHRONIZER.CPP:

```
// SYNCHRONIZER.CPP - Implementation file for your Internet Server
//    Synchronizer Extension

#include "stdafx.h"
#include "Synchronizer.h"
#include "isapithread.h"
#include "cominit.h"
#include <string>

////////////////////////////////////////////////////////////////////
// The one and only CWinApp object
// NOTE: You may remove this object if you alter your project to no
// longer use MFC in a DLL.

CWinApp theApp;

////////////////////////////////////////////////////////////////////
// command-parsing map

BEGIN_PARSE_MAP(CSynchronizerExtension, CHttpServer)
        // TODO: insert your ON_PARSE_COMMAND() and
        // ON_PARSE_COMMAND_PARAMS() here to hook up your commands.
        // For example:

        ON_PARSE_COMMAND(Default, CSynchronizerExtension, ITS_EMPTY)
        ON_PARSE_COMMAND(BroadcastVideoEvents, CSynchronizerExtension,
ITS_PSTR ITS_PSTR ITS_I4 ITS_PSTR ITS_PSTR ITS_PSTR)
        ON_PARSE_COMMAND(ServerInfo, CSynchronizerExtension, ITS_PSTR
ITS_PSTR)
        DEFAULT_PARSE_COMMAND(Default, CSynchronizerExtension)
END_PARSE_MAP(CSynchronizerExtension)


////////////////////////////////////////////////////////////////////
// The one and only CSynchronizerExtension object

CSynchronizerExtension theExtension;


////////////////////////////////////////////////////////////////////
// CSynchronizerExtension implementation

CSynchronizerExtension::CSynchronizerExtension()
{
```

```
        m_pisapithread = NULL;
}

CSynchronizerExtension::~CSynchronizerExtension()
{
        delete m_pisapithread;
        m_pisapithread = NULL;
}
static HINSTANCE g_hInstance;

HINSTANCE AFXISAPI GetResourceHandle()
{
        return g_hInstance;
}

BOOL CSynchronizerExtension::GetExtensionVersion(HSE_VERSION_INFO* pVer)
{
        // Call default implementation for initialization
        CHttpServer::GetExtensionVersion(pVer);

        // Load description string
        TCHAR sz[HSE_MAX_EXT_DLL_NAME_LEN+1];
        g_hInstance = AfxGetResourceHandle();
        ISAPIVERIFY(::LoadString(AfxGetResourceHandle(),
                        IDS_SERVER, sz, HSE_MAX_EXT_DLL_NAME_LEN));
        _tcscpy(pVer->lpszExtensionDesc, sz);
        BOOL bret = TRUE;
        try
        {
                m_pisapithread = new isapithread();
                m_pisapithread->start(NULL);
        }
        catch(...)
        {
                bret = FALSE;
        }
        return bret;
}

BOOL CSynchronizerExtension::TerminateExtension(DWORD dwFlags)
{
        // extension is being terminated
        //TODO: Clean up any per-instance resources
        if (m_pisapithread)
                m_pisapithread->stop();
        delete m_pisapithread;
```

```
        m_pisapithread = NULL;
        return TRUE;
}

////////////////////////////////////////////////////////////////
// CSynchronizerExtension command handlers

void CSynchronizerExtension::Default(CHttpServerContext* pCtxt)
{
        StartContent(pCtxt);
        WriteTitle(pCtxt);

        *pCtxt << _T("Invalid function call...");

        EndContent(pCtxt);
}

void CSynchronizerExtension::ServerInfo(CHttpServerContext* pCtxt,
                                                                LPCTSTR
lpDiskId,
                                                                LPCTSTR
nLocationID)
{
        StartContent(pCtxt);

        std::string locationId;
        std::string theDiskId = lpDiskId;
        locationId = nLocationID;

        if (m_pisapithread)
        {
                m_pisapithread->getBLayerInfo(theDiskId,locationId,pCtxt);
        }
        EndContent(pCtxt);
}

void CSynchronizerExtension::BroadcastVideoEvents(CHttpServerContext* pCtxt,

LPTSTR lpUserName,

LPCTSTR lpDiskId,

long nLocationID,

LPCTSTR lpBCA,
```

```
    LPCTSTR lpProcessorID,

    LPCTSTR lpDecoderID)
{

        if (m_pisapithread)
        {
                try
                {
                        m_pisapithread->addrequest(new request(pCtxt->m_pECB,
                                const_cast<char *>(lpDiskId),
                                nLocationID,
                                const_cast<char *>(lpDecoderID),
                                const_cast<char *>(lpUserName),
                                const_cast<char *>(lpBCA)));

                        pCtxt->m_dwStatusCode = HSE_STATUS_PENDING;
                        pCtxt->m_bSendHeaders = FALSE;
                }
                catch(IAUserException *piauserexc)
                {
                        *pCtxt << piauserexc->operator const char *();
                        delete piauserexc;
                }
        }
        else
        {
                *pCtxt << _T("Please comeback later...");
        }


}

// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CSynchronizerExtension, CHttpServer)
        //{{AFX_MSG_MAP(CSynchronizerExtension)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0


//////////////////////////////////////////////////////////////////////
// If your extension will not use MFC, you'll need this code to make
```

```
// sure the extension objects can find the resource handle for the
// module.  If you convert your extension to not be dependent on MFC,
// remove the comments arounn the following AfxGetResourceHandle()
// and DllMain() functions, as well as the g_hInstance global.

/****

static HINSTANCE g_hInstance;

HINSTANCE AFXISAPI AfxGetResourceHandle()
{
        return g_hInstance;
}

BOOL WINAPI DllMain(HINSTANCE hInst, ULONG ulReason,
                                        LPVOID lpReserved)
{
        if (ulReason == DLL_PROCESS_ATTACH)
        {
                g_hInstance = hInst;
        }

        return TRUE;
}

****/
```

ISAPITHREAD.H:

```
// isapithread.h: interface for the isapithread class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_ISAPITHREAD_H__F39740E6_355C_4103_BD65_2E65078E4D6E__I
NCLUDED_)
#define
AFX_ISAPITHREAD_H__F39740E6_355C_4103_BD65_2E65078E4D6E__INCLUDE
D_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "threadfunc.h"
#include "layersink.h"
#include <map>
#include <vector>
class CHTTPServerContext;

typedef std::map<std::string,layerSink * > mapsink;
typedef std::vector<request *> request_vector;
class isapithread : public threadFunctor
{
public:
        isapithread();
        virtual ~isapithread();

        void addrequest(request *prequest);
        void getBLayerInfo(std::string &diskID,std::string
&locationID,CHttpServerContext* pCtxt);
protected:
        void sendTime(CHttpServerContext * pCtxt, long title, long chapter, long
lapsedTime, long eventLength, CTime eventTime, CTime serverTime);
        void processrequest(request *prequest);

        void handler();
        HRESULT createfactory();

        CComPtr<ILayerFactory> m_ifactory;
        std::string m_dcomserver;
        mapsink      m_map;
        request_vector m_requestvector;
```

```
private:
        mutex m_mutex;

};

#endif //
!defined(AFX_ISAPITHREAD_H__F39740E6_355C_4103_BD65_2E65078E4D6E__I
NCLUDED_)
```

ISAPITHREAD.CPP:

```
// isapithread.cpp: implementation of the isapithread class.
//
////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "isapithread.h"
#include "cominit.h"
#include <comdef.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////

#define THREAD_KEY _T("SOFTWARE\\Interactual Technologies\\ServerConfig")
isapithread::isapithread()
{
        CRegKey key;
        if (key.Open(HKEY_LOCAL_MACHINE, THREAD_KEY) ==
ERROR_SUCCESS)
        {
                char value[_MAX_PATH];
                ULONG size = _MAX_PATH ;

                ZeroMemory(value, size);
                m_dcomserver.empty();
                try
                {
                        if (key.QueryValue(value,_T("DCOMServer"),&size) ==
ERROR_SUCCESS)
                        {
                                m_dcomserver = value;
                                if (m_dcomserver == "(local)")
                                        m_dcomserver = "";
                        }
                }
                catch(...)
                {
                }
```

```
        }
        else if (key.Create(HKEY_LOCAL_MACHINE, THREAD_KEY) ==
ERROR_SUCCESS)
        {
                key.SetValue("(local)",_T("DCOMServer"));
                m_dcomserver = "";
        }
}

isapithread::~isapithread()
{
}


void isapithread::handler()
{
        try
        {

                HRESULT hr = E_FAIL;
                m_mutex.lock();
                // need to create factory;
                hr = createfactory();
                m_mutex.unlock();

                if (FAILED(hr))
                {
                        throw(new COMException(hr));
                }

                while (1)
                {
                        // Copy the request vector so that we don't have to lock the mutex
again.
                        m_mutex.lock();
                        request_vector requestListCopy = m_requestvector;
                        m_requestvector.clear();
                        m_mutex.unlock();

                        // process each request in the queue
                        for (int i = requestListCopy.size() - 1; i >= 0; i--)
                        {
                                processrequest(requestListCopy[i]);
                                requestListCopy.pop_back();
                        }
```

```
                    checkCancel();

                    m_mutex.lock();
                    // do the processing
                    mapsink::iterator it = m_map.begin();
                    for (;it != m_map.end(); it++)
                    {
                        layerSink *pSink = (*it).second;
                        if (pSink && pSink->status() == layerSink::DEAD)
                        {
                            (*it).second = NULL;
                            pSink->clear();
                            pSink->Release();
                        }
                    }
                    m_mutex.unlock();
                    Sleep(500);

                }
            }
            catch(COMException *comException)
            {
                delete comException;
            }
            catch(...)
            {
            }

}

HRESULT isapithread::createfactory()
{
  COSERVERINFO cos ;
      COAUTHINFO athn;
  WCHAR wsz [MAX_PATH];
      memset(&cos, 0, sizeof(COSERVERINFO));
      HRESULT hr = E_FAIL;
      if (m_dcomserver.size() != 0)
      {
          MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED,
m_dcomserver.c_str(), -1,
              wsz, MAX_PATH);
          cos.pwszName = wsz;
          cos.pAuthInfo = &athn;
```

```
athn.pAuthIdentityData = (COAUTHIDENTITY *)new
SEC_WINNT_AUTH_IDENTITY;
athn.dwAuthnSvc = RPC_C_AUTHN_WINNT;
athn.dwAuthzSvc = RPC_C_AUTHZ_NONE;
athn.pwszServerPrincName = NULL;
athn.dwAuthnLevel = RPC_C_AUTHN_LEVEL_DEFAULT;
athn.dwCapabilities = EOAC_NONE;
athn.pAuthIdentityData->UserLength = _tcslen("dbmgr");
athn.pAuthIdentityData->DomainLength = _tcslen("IANT_DEVELOP");
athn.pAuthIdentityData->PasswordLength = _tcslen("ygm@neo");
athn.pAuthIdentityData->Domain = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->DomainLength + 1];
athn.pAuthIdentityData->User = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->UserLength + 1];
athn.pAuthIdentityData->Password = (unsigned short *)new TCHAR
[athn.pAuthIdentityData->PasswordLength + 1];
lstrcpy((TCHAR *)athn.pAuthIdentityData->Domain,
"IANT_DEVELOP");
lstrcpy((TCHAR *)athn.pAuthIdentityData->User, "dbmgr");
lstrcpy((TCHAR *)athn.pAuthIdentityData->Password, "ygm@neo");

}
MULTI_QI mqi;

mqi.pIID = &IID_ILayerFactory;
mqi.pItf = NULL;
mqi.hr = E_FAIL;

hr = CoCreateInstanceEx(CLSID_LayerFactory,NULL,
CLSCTX_ALL,&cos,1,&mqi);
char msg[_MAX_PATH];

sprintf(msg, "hr = %d\n", hr);
::OutputDebugString(msg);
if (SUCCEEDED(hr))
{
    ::OutputDebugString("Factory creation succeeded\n");
    hr = mqi.hr;
    if (SUCCEEDED(hr))
    {
        m_ifactory = reinterpret_cast<ILayerFactory *>(mqi.pItf);
    }
}

return hr;
```

```
}

void isapithread::addrequest(request *prequest)
{
        layerSink::sendHeaders(prequest);
        m_mutex.lock();
        m_requestvector.push_back(prequest);
        m_mutex.unlock();
}

void isapithread::getBLayerInfo(std::string &diskID,std::string
&locationID,CHttpServerContext* pCtxt)
{
        mapsink::iterator theIterator;
        long ldata;
        long title, chapter, eventLength, lapsedTime;
        std::string reqstr = diskID + locationID;
        std::string lEventStart,lEventStop;
        std::string
chapterOut,titleOut,startTimeOut,eventLenOut,lapsedTimeOut,serverTimeOut;

        m_mutex.lock();
        theIterator = m_map.find(reqstr);
        layerSink *playerSink  = NULL;


  if(theIterator != m_map.end() && (*theIterator).second != NULL)   // is 0 - 9
  {
                playerSink = (*theIterator).second;
                IBusinessLayer* m_ilayer;
                m_ilayer = playerSink->getBLayer();
                if (m_ilayer)
                {
                        CTime eventTime, serverTime;
                        m_ilayer->get_chapterProperties(&chapter);
                        m_ilayer->get_titleProperties(&title);
                        m_ilayer->get_startEvent(&ldata);
                        eventTime = ldata;
                        m_ilayer->get_eventLength(&eventLength);
                        m_ilayer->get_lapsedTime(&lapsedTime);
                        m_ilayer->get_serverTime(&ldata);
                        serverTime = ldata;

        sendTime(pCtxt,title,chapter,lapsedTime,eventLength,eventTime,serverTime);

                }
```

```
        }
        else if (m_ifactory)
        {
                long eventTime;
                long serverTime;

                m_ifactory->getScheduledTime(_bstr_t(diskID.c_str()),
_bstr_t(locationID.c_str()), &eventTime,&serverTime,&eventLength);
                sendTime(pCtxt,0,0,0,eventLength,eventTime,serverTime);
        }
        else
        {
                *pCtxt << _T("_INTERACTUAL_ERROR");
#ifdef _TRACE
                std::string error = "Error occured sending time info: location id = " +
locationID;
                ::OutputDebugString(error.c_str());
#endif
        }
        m_mutex.unlock();

}

void isapithread::sendTime(CHttpServerContext *pCtxt, long title, long chapter, long
lapsedTime, long eventLength, CTime eventTime, CTime serverTime)
{
                *pCtxt << (long int)chapter;
                *pCtxt << "\1";
                *pCtxt << (long int)title;
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%S");
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%M");
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%H");
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%d");
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%m");
                *pCtxt << "\1";
                *pCtxt << eventTime.Format("%Y");
                *pCtxt << "\1";
                *pCtxt << (long int)eventLength;
                *pCtxt << "\1";
                *pCtxt << (long int)lapsedTime;
                *pCtxt << "\1";
```

```
                *pCtxt << serverTime.Format("%S");
                *pCtxt << "\1";
                *pCtxt << serverTime.Format("%M");
                *pCtxt << "\1";
                *pCtxt << serverTime.Format("%H");
                *pCtxt << "\1";
                *pCtxt << serverTime.Format("%d");
                *pCtxt << "\1";
                *pCtxt << serverTime.Format("%m");
                *pCtxt << "\1";
                *pCtxt << serverTime.Format("%Y");
                *pCtxt << "\1";

}

void isapithread::processrequest(request *prequest)
{
        mapsink::iterator theIterator;
        std::string reqstr = prequest->get_diskid() + prequest->get_locationid();
        theIterator = m_map.find(reqstr);
        layerSink *playerSink  = NULL;
        try
        {
                if(theIterator != m_map.end() && (*theIterator).second != NULL)    // is 0
- 9
                {
                        playerSink = (*theIterator).second;
                        if (playerSink && playerSink->status() != layerSink::DEAD)
                                playerSink->addRequest(prequest);
                        else
                                throw new IAUserException("event has ended");

                }
                else
                {
                        playerSink  = createLayer();
                        playerSink->construct(prequest, m_ifactory, m_dcomserver);
                        m_map[reqstr] =  playerSink;
                }
        }
        catch(IAUserException *userException)
        {
                if (playerSink)
                        playerSink->Release();
                playerSink = NULL;
                m_map[reqstr] = 0;
```

```
                std::string msg = userException->operator const char *();
                layerSink::formatdata(msg);
                if (layerSink::writedata(prequest, msg))
                        layerSink::closeconnection(prequest);
                delete prequest;
                delete userException;

        }
        catch(COMException * comException)
        {
                if (playerSink)
                        playerSink->Release();
                playerSink = NULL;
                m_map[reqstr] = 0;
                delete comException;
                std::string msg;
                layerSink::formatdata(msg);
                if (layerSink::writedata(prequest, msg))
                        layerSink::closeconnection(prequest);
                delete prequest;
        }
        catch(...)
        {
                if (playerSink)
                        playerSink->Release();
                playerSink = NULL;
                m_map[reqstr] = 0;
                std::string msg;
                layerSink::formatdata(msg);
                if (layerSink::writedata(prequest, msg))
                        layerSink::closeconnection(prequest);
                delete prequest;
        }
}
```

THREADFUNCTOR.H:

```
// threadImpl.h: interface for the threadImpl class.
//
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__
INCLUDED_)
#define
AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__INCLU
DED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// main isapi thread
#include <process.h>
#include <tchar.h>
#include <map>
#include <vector>
typedef void (_cdecl * threadFunc)(void *);

#include "cominit.h"

//////////////////////////////////////////////////////////////////////
////////////////// application event log//////////////////////////////
class appEventLog
{
public :
        appEventLog(const char * eventName = NULL,
                                const char *srcName = _T("Application"));
        ~appEventLog();

        bool appReportErrorEvent(const char * msg,
                                                        WORD type =
EVENTLOG_ERROR_TYPE,
                                                        WORD eventId = 0,
                                                        WORD category = 7) const;

protected:
        HANDLE m_eventLogHandle;
};
```

```
class ExceptionCanceled : public COMException
{
public:
        ExceptionCanceled() : COMException(S_OK)
        {}

};

//////////////////////////////////////////////////////////////
////////////////// mutex                    ////////////////////////////

class mutex
{
public:
        mutex(LPSECURITY_ATTRIBUTES pattr = NULL, bool bOwned = false,  char
* name = NULL)
        {       m_mutex = CreateMutex(pattr, bOwned, name); m_locked = false;}
        ~mutex()
        { if (m_mutex)          CloseHandle(m_mutex);       m_mutex = NULL;    }
        bool isLocked()
        {       return m_locked;        }
        void lock()
        { ::WaitForSingleObject(m_mutex, INFINITE); m_locked = true;}
        void unlock()
        {::ReleaseMutex(m_mutex); m_locked = false;}

protected:
        HANDLE m_mutex;
        bool   m_locked;

};

//////////////////////////////////////////////////////////////
////////////////// threadFunctor            ////////////////////////////

class threadFunctor
{
public:
        threadFunctor();
        virtual ~threadFunctor();

        void set_event_log(appEventLog * eventlog)
        {       m_eventlog = eventlog;          }
```

```
        void writeevent(const char * msg, unsigned int eventId = 0, WORD type =
    EVENTLOG_ERROR_TYPE)
        { if (m_eventlog) m_eventlog->appReportErrorEvent(msg, type, eventId); }
        virtual void start(void *pParam,bool bSynch = true);
#ifndef _AFX
private:
#endif
        virtual void startWin(void *pParam,bool bSynch = true);
#ifndef _AFX
public:
#endif

        virtual void signal();
        void stop();
   HANDLE threadHandle();
   DWORD threadId();

protected:
        void checkCancel();
        virtual void startThread();

   virtual void prehandler();
        virtual void handler();
   virtual void posthandler();

        HANDLE m_threadhandle;
        HANDLE m_hEvent;  // event to terminate thread;
        void * m_tParam;
        bool  m_sync;
        appEventLog * m_eventlog;
   DWORD m_dwThreadId;
        friend void _cdecl threadfunction(void *pParam);
        friend UINT mfcthreadfunc( void *pparam );
};

////////////////////////////////////////////////////////////////
////////////////// threadPool                  ////////////////////////////

typedef std::map<int, class threadFunctor *> threadMap;

class threadPool
{
public:
        threadPool();
        virtual ~threadPool();
```

```
public:
        void signal();
    void set_event_log(appEventLog * eventlog);
        unsigned long wait(unsigned long timeout);
        void execute(int type,void *param);
        void remove(int type);
        void add(int, threadFunctor * pthread);
        void execute();
        threadFunctor * functor(int type);


protected:
        threadMap     m_threadMap;
};



#endif //
!defined(AFX_THREADIMPL_H__502CBBDA_69BE_42A9_863A_371BB377984C__
INCLUDED_)
```

THREADFUNCTOR.CPP:

```
// threadImpl.cpp: implementation of the threadImpl class.
//
////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include <process.h>
#include "threadfunc.h"

#ifdef _AFX
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif // _DEBUG
#endif //_AFX


////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////

threadFunctor::threadFunctor()
{
        m_tParam = NULL;
        m_threadhandle = NULL;
        m_hEvent = NULL;
    m_dwThreadId = 0;
        m_sync = false;
        m_eventlog = NULL;
}

threadFunctor::~threadFunctor()
{
        if (m_threadhandle)
                CloseHandle(m_threadhandle);
        if (m_hEvent)
                CloseHandle(m_hEvent);
}

void _cdecl threadfunction(void *pparam)
{
        threadFunctor* pfunc = reinterpret_cast<threadFunctor *> (pparam);
        if (pfunc)
        {
```

```
                pfunc->startThread();
        }
        _endthread();
}

UINT mfcthreadfunc( void *pparam )
{
        threadFunctor* pfunc = reinterpret_cast<threadFunctor *> (pparam);
        if (pfunc)
        {
                pfunc->startThread();
        }

        return 0;
}

void threadFunctor::startThread()
{
        COMInitalizer comInit;
   m_dwThreadId = ::GetCurrentThreadId();
        if (m_sync)
                ::SetEvent(m_hEvent);
        try
        {
      prehandler();
                handler();
      posthandler();
        }
        catch(...)
        {
        }
        m_threadhandle = NULL;
}

void threadFunctor::startWin(void *pParam,bool bSynch)
{
#ifndef _AFX
  assert("" == "Tried to invoke threadFunctor::startWin() without MFC");
#else // _AFX
        m_tParam = pParam;
        if (!m_hEvent)
                m_hEvent = CreateEvent(NULL,false,false, _T("_InterActual_EVENT"));
        m_sync = bSynch;
        m_threadhandle = (HANDLE)AfxBeginThread(mfcthreadfunc,this);
        if (m_hEvent && bSynch)
        {
```

```
                WaitForSingleObject(m_hEvent, INFINITE);
                ResetEvent(m_hEvent);
        }
        else if (bSynch)
        {
                throw new IAUserException("Event failed to create");
        }
#endif // _AFX
}

void threadFunctor::start(void *pParam,bool bSynch)
{
        m_tParam = pParam;
        m_sync = bSynch;
        if (!m_hEvent)
                m_hEvent = CreateEvent(NULL,false,false,NULL);
        m_threadhandle = (HANDLE)_beginthread(threadfunction,0, this);

        if (m_hEvent && bSynch)
        {
                WaitForSingleObject(m_hEvent, INFINITE);
                ResetEvent(m_hEvent);
        }
        else if (bSynch)
        {
                throw new IAUserException("Event failed to create");
        }
}


void threadFunctor::prehandler()
{
  // Do nothing
}

void threadFunctor::handler()
{
        DWORD result;
        while ((result = WaitForSingleObject(m_hEvent, 0)) != WAIT_OBJECT_0)
        {
                checkCancel();
        }
}
```

```
void threadFunctor::posthandler()
{
  // Do nothing
}

void threadFunctor::signal()
{
        if (m_hEvent)
                ::SetEvent(m_hEvent);
}

void threadFunctor::stop()
{
#ifndef _WIN32_DCOM
        ATLTRACE("thread %p is stopping", this);
#endif
        if (m_threadhandle)
        {
                signal();
                WaitForSingleObject(m_threadhandle, INFINITE);
                if (m_threadhandle)
                        CloseHandle(m_threadhandle);
        }
        CloseHandle(m_hEvent);
        m_threadhandle = NULL;
        m_hEvent = NULL;

}


void threadFunctor::checkCancel()
{
        DWORD result;
        if (m_hEvent)
        {
                result = WaitForSingleObject(m_hEvent, 0);
                switch(result)
                {
                case WAIT_ABANDONED:
                break;

                case WAIT_OBJECT_0:
                        throw new ExceptionCanceled();
                break;
```

```
                };
            }
        }

        HANDLE threadFunctor::threadHandle()
        {
          return m_threadhandle;
        }

        DWORD threadFunctor::threadId()
        {
          return m_dwThreadId; // Might not have been set if thread start not synched
        }

        ////////////////////////////////////////////////////////////////
        // threadPool Class
        ////////////////////////////////////////////////////////////////

        ////////////////////////////////////////////////////////////////
        // Construction/Destruction
        ////////////////////////////////////////////////////////////////

        threadPool::threadPool()
        {

        }

        threadPool::~threadPool()
        {
            threadMap::iterator it = m_threadMap.begin();

            for(;it != m_threadMap.end();it++)
            {
                threadFunctor * functor = (*it).second;
                if (functor)
                {
                    functor->stop();
                    delete functor;
                }
            }
            m_threadMap.clear();
        }

        void threadPool::add(int type, threadFunctor *pthread)
```

```
{
        threadMap::iterator it;
        it = m_threadMap.find(type);
        if (it == m_threadMap.end())
                m_threadMap[type] = pthread;
}

void threadPool::remove(int type)
{
        threadMap::iterator it = m_threadMap.find(type);
        if (it != m_threadMap.end() && (*it).second != NULL)
        {
                threadFunctor * functor = (*it).second;
                functor->stop();
                delete functor;
                m_threadMap[type] = NULL;
        }

}

void threadPool::execute(int type, void *param)
{
        threadMap::iterator it = m_threadMap.find(type);
        if (it != m_threadMap.end() && (*it).second != NULL)
        {
                (*it).second->start(param);
        }

}

void threadPool::set_event_log(appEventLog * eventlog)
{
        threadMap::iterator it = m_threadMap.begin();
        for  (;it != m_threadMap.end(); it++)
        {
          if ((*it).second != NULL)
                        (*it).second->set_event_log(eventlog);
        }
}

void threadPool::execute()
{
        threadMap::iterator it = m_threadMap.begin();
        for (;it != m_threadMap.end(); it++)
        {
```

```
                threadFunctor * functor = (*it).second;
                if (functor != NULL)
                {
                        functor->start(functor);
                        SleepEx(2, true);
                }

        }
}

threadFunctor * threadPool::functor(int type)
{
        threadFunctor * functor = NULL;
        threadMap::iterator it = m_threadMap.find(type);
        if (it != m_threadMap.end() && (*it).second != NULL)
        {
                functor = (*it).second;
        }
        return functor;
}

unsigned long threadPool::wait(unsigned long timeout)
{
        unsigned long lRet = -1l;
        HANDLE *handleMap = new HANDLE[m_threadMap.size()];
        for (int i = 0; i < m_threadMap.size(); i++)
        {
                handleMap[i] = m_threadMap[i]->threadHandle();
        }
        lRet = ::WaitForMultipleObjects(m_threadMap.size(), handleMap, true, timeout);
        delete handleMap;
        return lRet;
}

void threadPool::signal()
{
        for (int i = 0; i < m_threadMap.size(); i++)
        {
                m_threadMap[i]->signal();
        }

}

////////////////////////////////////////////////////////////////////////////////
```

```
////////////////// Application event log ////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////


const char *SubKey =
_T("System\\CurrentControlSet\\Services\\EventLog\\Application\\");

appEventLog::appEventLog(const char * eventName, const char * srcName)
{
        USES_CONVERSION;
        m_eventLogHandle = NULL;
        std::string subKey = std::string(SubKey) + srcName;
        CRegKey       regkey;

        regkey.Create(HKEY_LOCAL_MACHINE, subKey.c_str());

        m_eventLogHandle = RegisterEventSource(eventName, srcName);

        if (!m_eventLogHandle)
        {
                ATLTRACE("Could not register event source error = %d",
GetLastError());
        }
}

appEventLog::~appEventLog()
{
        if (m_eventLogHandle)
                DeregisterEventSource(m_eventLogHandle);
}

bool appEventLog::appReportErrorEvent(const char * msg,
                        WORD type /*= EVENTLOG_ERROR_TYPE*/,
                        WORD eventId /*= 0*/,
                        WORD category /*= 7*/) const
{
        bool bresult = false;
        if (m_eventLogHandle)
        {
#pragma warning(push)
#pragma warning(disable : 4800)
                bresult = ReportEvent(m_eventLogHandle, type, category, eventId,
NULL,
                        1, 0, (const char **)&msg , NULL);
#pragma warning(pop)
   }
```

```
        return bresult;
}
```

BUSINESSLAYER.H:

```
// BusinessLayer.h : Declaration of the CBusinessLayer

#ifndef __BUSINESSLAYER_H_
#define __BUSINESSLAYER_H_

#include "resource.h"       // main symbols

#include "ConfigurationManager.h"
#include "DBConnector.h"
#include <comdef.h>

#include <map>

#include <string>
#include "HiddenWnd.h"
#include "LayerImplCP.h"

#include "threadfunc.h"

class layerthread;

typedef std::map<long,long> decoderCapabilities;
/////////////////////////////////////////////////////////////////////////////
// CBusinessLayer
class ATL_NO_VTABLE CBusinessLayer :
        public CComObjectRootEx<CComMultiThreadModel>,
        public CComCoClass<CBusinessLayer, &CLSID_BusinessLayer>,
        public ISupportErrorInfo,
        public IConnectionPointContainerImpl<CBusinessLayer>,
        public IBusinessLayer,
        public CProxy_IBusinessLayerEvents< CBusinessLayer >
{
protected:
        std::string m_diskid;
        std::string m_location;
        _bstr_t  m_diskID;
        _bstr_t  m_locationID;
        CTime m_serverTime;
        CTime m_eventLength;
        CTime m_lapsedTime;
        CTime m_startEvent;
        CTime m_stopEvent;
        long threshold;
        short m_hostType;
```

```
        long m_title;
        long  m_chapter;
        IDB_Connector *m_pIDBConnect;
        ICConfigMgrImpl *m_pICConfigMgrImpl;
        layerthread * m_pthread;
        bool m_firstTime;
        decoderCapabilities m_capabilities;


protected:
        void ChkValidEvent();
        HRESULT GetNextPair(long *theTime, long *nTitle, long * nChapter,  BSTR
*chapterCmnd);
        void sendCommand(BSTR chapterCmnd);
        void endSession(BSTR  szMsg);
        void updateTime(LONG  time,LONG nTitle);
        _bstr_t firstdvdCmd();
        void put_serverTime(/*[in]*/ long serverTime, long title, long chapter, long
lapsedTime, long length);
        mutex   m_timeLock;

public:
        CBusinessLayer()
        {
                m_pthread = NULL;
                m_pIDBConnect = NULL;
                m_pICConfigMgrImpl = NULL;
                m_diskid.empty();
                m_location.empty();
        }

        HRESULT FinalConstruct();
        void FinalRelease();

DECLARE_REGISTRY_RESOURCEID(IDR_BUSINESSLAYER)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CBusinessLayer)
        COM_INTERFACE_ENTRY(IBusinessLayer)
        COM_INTERFACE_ENTRY(ISupportErrorInfo)
        COM_INTERFACE_ENTRY(IConnectionPointContainer)
        COM_INTERFACE_ENTRY_IMPL(IConnectionPointContainer)
END_COM_MAP()

BEGIN_CONNECTION_POINT_MAP(CBusinessLayer)
```

```
CONNECTION_POINT_ENTRY(IID__IBusinessLayerEvents)
END_CONNECTION_POINT_MAP()


// ISupportsErrorInfo
        STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

// IBusinessLayer
        void ProcessEvent();

public:
        STDMETHOD(get_serverTime)(/*[out, retval]*/ long *pVal);
        STDMETHOD(get_titleProperties)(/*[out, retval]*/ long *pVal);
        STDMETHOD(get_chapterProperties)(/*[out, retval]*/ long *pVal);
        STDMETHOD(get_lapsedTime)(/*[out, retval]*/ long *pVal);
        STDMETHOD(get_eventLength)(/*[out, retval]*/ long *pVal);
        STDMETHOD(TranslateTimePlay)(/*[in]*/ long nDecoderType, /*[in]*/long
nTitle, /*[in]*/long nTime, /*[out, retval]*/ BSTR *szCmd);
        STDMETHOD(Initialize)();
        STDMETHOD(get_threshold)(/*[out, retval]*/ long *pVal);
        STDMETHOD(put_threshold)(/*[in]*/ long newVal);
        STDMETHOD(get_stopEvent)(/*[out, retval]*/ long *pVal);
        STDMETHOD(put_stopEvent)(/*[in]*/ long newVal);
        STDMETHOD(get_startEvent)(/*[out, retval]*/ long *pVal);
        STDMETHOD(put_startEvent)(/*[in]*/ long newVal);
        STDMETHOD(get_location)(/*[out, retval]*/ BSTR *pVal);
        STDMETHOD(put_location)(/*[in]*/ BSTR newVal);
        STDMETHOD(get_disk)(/*[out, retval]*/ BSTR *pVal);
        STDMETHOD(put_disk)(/*[in]*/ BSTR newVal);
friend class layerthread;
};

#endif //__BUSINESSLAYER_H_
```

BUSINESSLAYER.CPP:

```
// BusinessLayer.cpp : Implementation of CBusinessLayer
#include "stdafx.h"
#include "layerimpl.h"
#include "BusinessLayer.h"
#include "process.h"

class layerthread : public threadFunctor
{
public:
	void startThread()
	{
		COMInitalizer init;
		try
		{
			handler();
		}
		catch(...)
		{
		}
		m_threadhandle = NULL;
	}
	void handler()
	{
		long threshold;
		long nextChapterTime;
		CTime startEvent;
		CTime stopEvent;
		CTime time;
		long eventLength;
		char msg[256];

		::OutputDebugString("\n Loop started\n");
		CBusinessLayer *pLayer = reinterpret_cast<CBusinessLayer
*>(m_tParam);
		if (pLayer)
		{
			try
			{

				time = CTime::GetCurrentTime() ;
				pLayer->get_threshold(&threshold);
				long timeData;
				pLayer->get_startEvent(&timeData);
				startEvent = timeData;
```

```
                                pLayer->get_stopEvent(&timeData);
                                stopEvent = timeData;

                                eventLength = CTimeSpan(stopEvent -
 5  startEvent).GetTotalSeconds() * 1000;

                                sprintf(msg, "%s\n", startEvent.Format("%D:%H:%M"));
                                ::OutputDebugString(msg);
                                sprintf(msg, "%s\n", stopEvent.Format("%D:%H:%M"));
10                              ::OutputDebugString(msg);
                                pLayer->put_serverTime(time.GetTime(),-1, -1 ,0,
    eventLength);
                                if (startEvent < stopEvent &&
                                        (time + CTimeSpan(threshold)) >= startEvent)
15                              {
                                        //
                                        // - Determine if it is time to kick-off the event.
                                        // - If it is, stop the loop.
                                        // - If it is NOT, go to sleep and check after 500
20  milliseconds
                                        //
                                        _bstr_t dvdCmd = pLayer->firstdvdCmd();
                                        std::string debugmsg;
                                        debugmsg = "first dvd command" + dvdCmd + "\n";
25
                                        ::OutputDebugString(debugmsg.c_str());
                                        while (time < startEvent )
                                        {
                                                checkCancel();
30                                              Sleep(500);
                                                time = CTime::GetCurrentTime() ;
                                                pLayer->put_serverTime(time.GetTime(),-1,
    -1 ,0,  eventLength);
                                        }
35                                      if (time <= (startEvent + CTimeSpan(1)))
                                        {
                                                pLayer->sendCommand(dvdCmd);

                                                ::OutputDebugString("Process Event");
40                                      }

                                        while (time < stopEvent)
                                        {

45                                              BSTR command;
                                                CTimeSpan lapsedTime(0);
```

```
                                        long title, chapter;

                                        if (time > startEvent)
                                        {
                                                lapsedTime = time - startEvent;
                                        }

                                        if (SUCCEEDED(pLayer-
>GetNextPair(&nextChapterTime,&title, &chapter, &command)))
                                        {
                                                if (lapsedTime.GetTotalSeconds() <
nextChapterTime)
                                                {

        while(lapsedTime.GetTotalSeconds() <= nextChapterTime)
                                                        {
                                                                checkCancel();
                                                                Sleep(500);
                                                                time =
CTime::GetCurrentTime();

                                                                lapsedTime = time -
startEvent;
                                                                pLayer-
>updateTime(lapsedTime.GetTotalSeconds(),title);
                                                                pLayer-
>put_serverTime(time.GetTime(),title, chapter ,lapsedTime.GetTotalSeconds() * 1000,
eventLength);
                                                        }

                                                        pLayer-
>sendCommand(command);
                                                        Sleep(500);
                                                }
                                                time = CTime::GetCurrentTime() ;
                                                ::SysFreeString(command);
                                        }
                                        else
                                        {
                                                TRACE("no more entries in the
chapter table");
                                                break;
                                        }
                                }
                        }
```

```
                        ::OutputDebugString("End Session");
                        pLayer->endSession(NULL);
                }
                catch(ExceptionCanceled * pExcp)
                {
                        delete pExcp;
                        _bstr_t msg = "Cancellation occured";
                        pLayer->endSession(msg);
                }
                catch(...)
                {
                        _bstr_t msg = "Cancellation occured";
                        pLayer->endSession(msg);
                }
        }
    }

};

////////////////////////////////////////////////////////////////////////////
// CBusinessLayer

STDMETHODIMP CBusinessLayer::InterfaceSupportsErrorInfo(REFIID riid)
{
        static const IID* arr[] =
        {
                &IID_IBusinessLayer
        };
        for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
        {
                if (InlineIsEqualGUID(*arr[i],riid))
                        return S_OK;
        }
        return S_FALSE;
}

HRESULT CBusinessLayer::FinalConstruct()
{


        HRESULT hr = CoCreateInstance(CLSID_CConfigMgrImpl,
                                                        0,
                                                        CLSCTX_ALL,
                                                        IID_ICConfigMgrImpl,
```

```
(void**)&m_pICConfigMgrImpl);

        return hr;
}

void CBusinessLayer::FinalRelease()
{
        if (m_pthread)
                m_pthread->stop();
        delete m_pthread;
        if (m_pICConfigMgrImpl)
                m_pICConfigMgrImpl->Release();
        try
        {
                if (m_pIDBConnect)
                        m_pIDBConnect->Release();
        }
        catch(...)
        {
        }
}

void CBusinessLayer::ChkValidEvent()
{
        ::OutputDebugString("\n Check valid Event\n");
        m_firstTime = false;
        if (m_diskID.length() == 0)

                throw new IAUserException("Invalid Disk id");

        if (m_pICConfigMgrImpl)
        {
                std::string debugmsg;
                debugmsg = "disk id=" + m_diskID + "; location id = " + m_locationID +
"\n";
                ::OutputDebugString(debugmsg.c_str());
                m_pICConfigMgrImpl->put_diskID(m_diskID);               // Variable
used for search critera
                m_pICConfigMgrImpl->put_locationID(m_locationID); // Variable used
for search critera
                m_pICConfigMgrImpl->get_hostType(&m_hostType);

                if (m_hostType)
                {
                        //
```

```
                    // Create a DBConnector, store the pointer for future use.
                    // Store values from db.
                    //
                    ::OutputDebugString("\n Host type is checked\n");
                    HRESULT hr = S_OK;
                    if (!m_pIDBConnect)
                    {
                        hr = CoCreateInstance(CLSID_DB_Connector,
                                              0,
                                              CLSCTX_ALL,
                                              IID_IDB_Connector,
    (void**)&m_pIDBConnect);
                    }
                    if (SUCCEEDED(hr))
                    {
                        ::OutputDebugString("\n Initialize DB Connector\n");
                        m_pIDBConnect->put_diskID(m_diskID);          //
Variable used for search critera
                        m_pIDBConnect->put_locationID(m_locationID); //
Variable used for search critera
                        m_pIDBConnect->chkEvent();
                        BSTR data;
                        m_pIDBConnect->get_diskID(&data);
                        if (data)
                        {
                            m_diskID = data;
                            ::SysFreeString(data);
                        }
                        m_pIDBConnect->get_locationID(&data);
                        if (data)
                        {
                            m_locationID = data;
                            ::SysFreeString(data);
                        }
                        long time;
                        m_pIDBConnect->get_startEvent(&time);
                        m_startEvent = time;
                        m_pIDBConnect->get_stopEvent(&time);
                        m_stopEvent = time;
                        m_pIDBConnect->get_thresold(&threshold);
                        m_pIDBConnect->get_hostType(&m_hostType);

                        long * nDecoderArray;
                        long * nCapabilitiesArray ;
                        nDecoderArray = nCapabilitiesArray = NULL;
```

```
                        if (SUCCEEDED(m_pIDBConnect-
>decoderArray(&nDecoderArray, &nCapabilitiesArray)))
                        {
                                int i = 0;
                                while(nDecoderArray[i] != -1)
                                {
                                        m_capabilities[nDecoderArray[i]] =
nCapabilitiesArray[i];
                                        i++;
                                }
                                CoTaskMemFree(nDecoderArray);
                                CoTaskMemFree(nCapabilitiesArray);
                        }

                        ::OutputDebugString("\n Prepare to start thread\n");

                        m_pthread = new layerthread;
                        m_pthread->start(this,false);
                }
                else
                {
                        throw new COMException(hr);
                }

        }
        else
        {
                //
                // Create a Reference Connector, and store the pointer for future
use.
                // TBD
                //
        }
    }
}


STDMETHODIMP CBusinessLayer::get_disk(BSTR* pVal)
{
    // TODO: Add your implementation code here
    *pVal = m_diskID.copy( );
    return S_OK;
}

STDMETHODIMP CBusinessLayer::put_disk(BSTR newVal)
```

```
{
        // TODO: Add your implementation code here
        m_diskID = newVal;
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_location(BSTR* pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_diskID.copy( );
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_location(BSTR newVal)
{
        // TODO: Add your implementation code here
        m_locationID = newVal;

        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_startEvent(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_startEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_startEvent(long newVal)
{
        // TODO: Add your implementation code here
        time_t time = newVal;
        m_startEvent = time;
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_stopEvent(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = m_stopEvent.GetTime(); // m_pIDBConnect->get_startEvent(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_stopEvent(long newVal)
{
        // TODO: Add your implementation code here
```

```
        time_t time = newVal;
        m_stopEvent = time;
        return S_OK;
}

STDMETHODIMP CBusinessLayer::get_threshold(long *pVal)
{
        // TODO: Add your implementation code here
        *pVal = threshold; // m_pIDBConnect->get_thresold(pVal);
        return S_OK;
}

STDMETHODIMP CBusinessLayer::put_threshold(long newVal)
{
        // TODO: Add your implementation code here
        threshold = newVal;
        return S_OK;
}

HRESULT CBusinessLayer::GetNextPair(long *theTime, long *nTitle, long * nChapter,
BSTR *chapterCmnd)
{
        return m_pIDBConnect-
>get_NextChapter(theTime,nTitle,nChapter,chapterCmnd);
}



_bstr_t CBusinessLayer::firstdvdCmd()
{
        //
        // Execute the first DVD Command
        //
        BSTR msg = NULL;
        _bstr_t dvdMsg;
        m_pIDBConnect->get_initialDVDCommand(&msg);
        if (msg)
                dvdMsg = msg;
        return dvdMsg;
}

void CBusinessLayer::sendCommand(BSTR szMsg)
{
        Fire_sendCommand(szMsg);
```

```
}

void CBusinessLayer::endSession(BSTR szMsg)
{
    Fire_endSession(szMsg);

}


void CBusinessLayer::updateTime(LONG time, long nTitle)
{
    Fire_updatetime(time,nTitle);
}

STDMETHODIMP CBusinessLayer::Initialize()
{
    HRESULT hr = S_OK;
    try
    {
        ChkValidEvent();
    }
    catch(IAUserException *pexcpt)
    {
        delete pexcpt;
        hr = E_FAIL;
        _bstr_t msg = "USER exception occured\n";
        Fire_endSession(msg);
    }
    catch(COMException * pcomexcpt)
    {
        hr = pcomexcpt->operator HRESULT();
        _bstr_t msg = "COM exception occured\n";
        Fire_endSession(msg);
    }
    catch(...)
    {
        _bstr_t msg = "Unknown exception occured\n";
        Fire_endSession(msg);

    }

    return hr;
}

STDMETHODIMP CBusinessLayer::TranslateTimePlay(long nDecoderType, long
nTitle, long nTime, BSTR *szCmd)
```

```
{
        // TODO: Add your implementation code here
        HRESULT hr = E_FAIL;
        decoderCapabilities::iterator it = m_capabilities.find(nDecoderType);
        if (it != m_capabilities.end())
        {
                if ((*it).second == 0)
                {
                        char translate[_MAX_PATH];
                        sprintf(translate, "tmp:%d:%d", nTitle, nTime * 1000);
                        *szCmd = _bstr_t(translate).copy();
                        hr = S_OK;
                }

        }

        return hr;
}

STDMETHODIMP CBusinessLayer::get_eventLengtb(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_eventLength.GetTime();
        m_timeLock.unlock();
        return S_OK;
}


STDMETHODIMP CBusinessLayer::get_lapsedTime(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_lapsedTime.GetTime();
        m_timeLock.unlock();

        return S_OK;
}


STDMETHODIMP CBusinessLayer::get_chapterProperties(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_chapter;
        m_timeLock.unlock();
```

```
        return S_OK;
}


STDMETHODIMP CBusinessLayer::get_titleProperties(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_title;
        m_timeLock.unlock();
        return S_OK;
}


STDMETHODIMP CBusinessLayer::get_serverTime(long *pVal)
{
        // TODO: Add your implementation code here
        m_timeLock.lock();
        *pVal = m_serverTime.GetTime();
        if (*pVal == 0)
        {
                char msg[1024];

                sprintf(msg, "title = %d,  chapter = %d, location = %s\n",m_title,
m_chapter, m_locationID.operator char *());
        }
        m_timeLock.unlock();

        return S_OK;
}
void CBusinessLayer::put_serverTime(/*[in]*/ long serverTime, long title, long chapter,
long lapsedTime, long length)
{
        m_timeLock.lock();
        m_serverTime = serverTime;
        m_title = title;
        m_chapter = chapter;
        m_lapsedTime = lapsedTime;
        m_eventLength = length;
        m_timeLock.unlock();
}
```